Bone Deformity Detection From X-Ray Images

Members

Sara Naziri

Taha Asqari

Peyman Afshari

CE.Guilan

مقدمه

در این برنامه هدف ما بررسی و شناسایی وجود یا عدم وجود نا هنجاری های موجود درعکس استخوان بازو با استفاده از مدل Inception V 3(Binary_Classification) میباشد.ما بااستفاده از این مدل به دقت ۷۹.۸٪ رسیدیم.

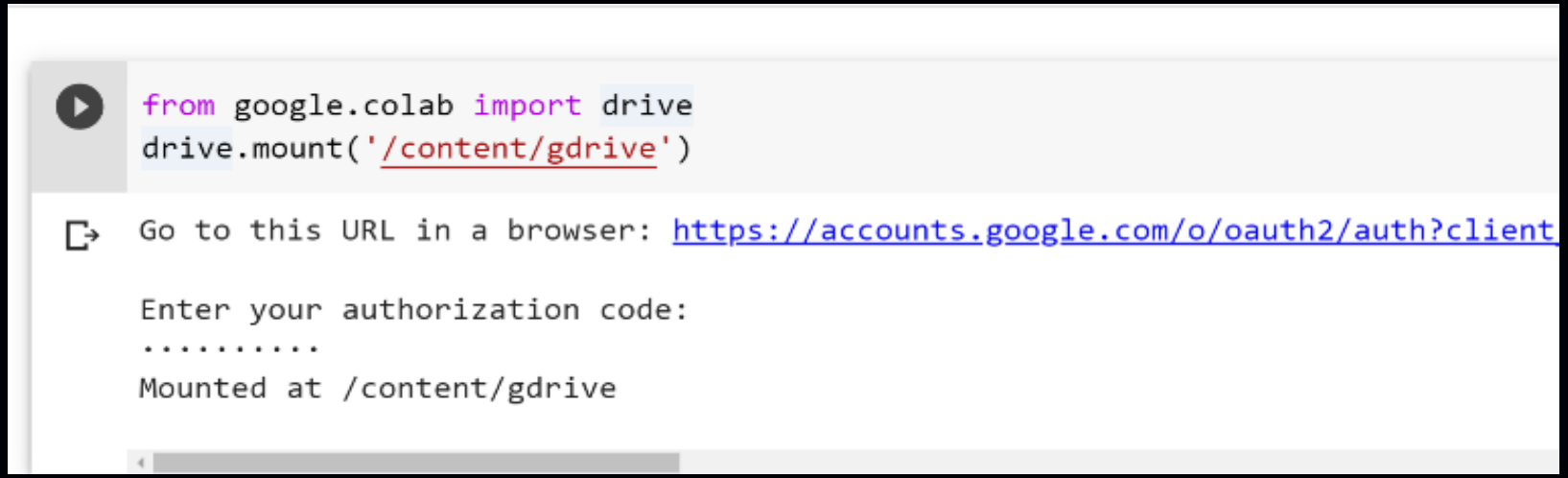
ما با استفاده از داده های موجود در دیتا ست مورا که توسط دانشگاه استنفورد جمع آوری شده شروع به یادگیری از تصاویر کرده سپس با استفاده از این یادگیری به بررسی و شناسایی ناهنجاری های موجود در تصاویر دیگر میکنیم.

MURA: Large Dataset for Abnormality Detection in Musculoskeletal Radiographs

- We introduce MURA, a large dataset of musculoskeletal radiographs containing 40,561 images from 14,863 studies, where each study is manually labeled by radiologists as either normal or abnormal. To evaluate models robustly and to get an estimate of radiologist performance, we collect additional labels from six board-certified Stanford radiologists on the test set, consisting of 207 musculoskeletal studies. On this test set, the majority vote of a group of three radiologists serves as gold standard. We train a 169-layer DenseNet baseline model to detect and localize abnormalities. Our model achieves an AUROC of 0.929, with an operating point of 0.815 sensitivity and 0.887 specificity. We compare our model and radiologists on the Cohen's kappa statistic, which expresses the agreement of our model and of each radiologist with the gold standard. Model performance is comparable to the best radiologist performance in detecting abnormalities on finger and wrist studies. However, model performance is lower than best radiologist performance in detecting abnormalities on elbow, forearm, hand, humerus, and shoulder studies. We believe that the task is a good challenge for future research. To encourage advances, we have made our dataset freely available at https://stanfordmlgroup.github.io/competitions/mura/

مرحله اول

دریافت اجازه دسترسی به گوگل درایو



مرحله دوم

فتچ کردن دیتا از Kaggle روی Google Colab

```
!kaggle datasets download -d cjinny/mura-v11
```

```
Downloading mura-v11.zip to /content/gdrive/My Drive/Kaggle
100% 3.14G/3.14G [00:55<00:00, 39.1MB/s]
100% 3.14G/3.14G [00:56<00:00, 59.5MB/s]
```

```
%cd /content/gdrive/My Drive/Kaggle
!pwd
!ls
```

```
/content/gdrive/My Drive/Kaggle
/content/gdrive/My Drive/Kaggle
kaggle.json
```

!pwd : چک کردن بودن در مسیر وارد شده(تایید صحت شناسایی پوشه)

!ls : نشان دادن محتویات پوشه وارد شده

مرحله سوم

Unzipping

شروع به استخراج داده های موجود(تصاویر) از فایل زیپ در داخل پوشه kaggle میکنیم(این پروسه زمان بر است)

```
!unzip \*.zip
# !unzip \*.zip  && rm *.zip
```

```
Streaming output truncated to the last 5000 lines.
  inflating: MURA-v1.1/train/XR_WRIST/patient08489/study1_negative/image4.png
  inflating: MURA-v1.1/train/XR_WRIST/patient08490/study1_negative/image1.png
  inflating: MURA-v1.1/train/XR_WRIST/patient08490/study1_negative/image2.png
  inflating: MURA-v1.1/train/XR_WRIST/patient08490/study1_negative/image3.png
  inflating: MURA-v1.1/train/XR_WRIST/patient08491/study1_negative/image1.png
  inflating: MURA-v1.1/train/XR_WRIST/patient08491/study1_negative/image2.png
  inflating: MURA-v1.1/train/XR_WRIST/patient08492/study1_negative/image1.png
```

مرحله چهارم

استفاده از Cross با بکند tensorflow

وارد کردن کتابخانه های موردنیاز برای برنامه

```
###### IMPORTs ######
import cv2
import numpy as np
import tensorflow as tf
import tensorflow_hub as hub
import tensorflow_datasets as tfds
import os, glob, shutil
from tqdm.notebook import tqdm
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from PIL import Image
```

مرحله پنجم

پیش پردازش دیتا(دیتا را براساس مدلی که باید بخواند پیش پردازش میکنیم)

مورا شامل هفت دسته است و ما از دسته استخوان بازو استفاده کردیم چرا که دسته کوچکتری نسبت به بقیه است و سرعت انجام عملیات بالاتر است.همچنین میتوانیم دیگر دسته ها را نیز اضافه کنیم.

```
train_files = glob.glob("/content/gdrive/My Drive/Kaggle/MURA-v1.1/train/XR_HUMERUS/**/**/*.png")
valid_files = glob.glob("/content/gdrive/My Drive/Kaggle/MURA-v1.1/valid/XR_HUMERUS/**/**/*.png")
print(len(train_files))
print(len(valid_files))
```

```
1272
288
```

Train_Files

Valid_Files

```
# !rm -rf ./prep
# !ls ./prep
```

Trainها را داخل این پوشه میریزیم.
از این دو دستور زمانی استفاده میکنیم که دچار کرش بشویم و به جای اینکه از اول شروع شود به اینجا می آییم.
دستور !rm – rf ./prep را زمانی اجرا میکنیم که دیتا آسیب ببیند و بخواهیم پوشه prep را پاک کنیم.

مرحله ششم

ریختن دیتا داخل پوشه prep

دو پوشه با نام train,valid میسازد(:train_folderعکس های یادگیری شبکه , :valid_foldrبرای validکردن شبکه استفاده میشود)

که هرکدام شامل دو پوشه positive,negative میشوند.(: positiveنشان وجود ناهنجاری, negative عدم وجود ناهنجاری)

```
### Creat preprocessing floder for train ###
if os.path.isdir('./prep/train/negative/'):
  pass
else:
  os.makedirs('./prep/train/negative/')
  os.makedirs('./prep/train/positive/')
!ls
!pwd
```

```
prep                    train_labeled_studies.csv  valid_labeled_studies.csv
train                   valid
train_image_paths.csv   valid_image_paths.csv
/content/gdrive/My Drive/Kaggle/MURA-v1.1
```

```
####preprocssing data###
neg_c = 0
pos_c = 0
for file_name in tqdm(train_files, total=len(train_files)):
  label = file_name.split('/')[-2]
  if "negative" in label:
    shutil.copy(file_name, f'./prep/train/negative/image_{str(neg_c)}.png')
    neg_c += 1
  elif "positive" in label:
    shutil.copy(file_name, f'./prep/train/positive/image_{str(pos_c)}.png')
    pos_c += 1
```

```
100% 1272/1272 [12:23<00:00, 1.71it/s]
```

در اینجا برای جلو گیری از مشکل تشابه فایل ها خودمان به آنها شماره ای به اسم neg-c و pos_c اضافه می کنیم.
هم برای Train_Files و هم برای Valid_Files این کار را میکنیم.

```
!pwd
!ls /content/gdrive/My\ Drive/Kaggle/MURA-v1.1/prep/train
```

```
/content/gdrive/My Drive/Kaggle/MURA-v1.1
negative  positive
```

این سلول برای حاصل شدن اطمینان از وجود مسیر درست فایل ها است.

مرحله هفتم

مسیر دهی برای شروع آموزش(train)

در لاین آخر عددهای پرینت شده را که در سلول بالا بخش پیدا کردن تعداد آدرس بود , به متغیرها وارد می کنیم.

```
train_dir = "/content/gdrive/My Drive/Kaggle/MURA-v1.1/prep/train/"
valid_dir = "/content/gdrive/My Drive/Kaggle/MURA-v1.1/prep/valid/"
train_pos = "/content/gdrive/My Drive/Kaggle/MURA-v1.1/prep/train/positive/"
train_neg = "/content/gdrive/My Drive/Kaggle/MURA-v1.1/prep/train/negative/"
val_pos = "/content/gdrive/My Drive/Kaggle/MURA-v1.1/prep/valid/positive/"
val_neg = "/content/gdrive/My Drive/Kaggle/MURA-v1.1/prep/valid/negative/"
total_train, total_val = (1272), (288)
```

Total_valid,Total_train: **کل مقادیر** positive,negative **است.**

در مدل ما مقادیر positive **مهم است (وجود ناهنجاری ها).**

در این سلول جزئیات تعداد تصاویر را در پایین سلول وجود دارد.

مرحله هشتم

تنظیم پارامتر های اصلی

```
IMG_SIZE = 150
batch_size = 16
epochs = 30
```

در اینجا مقادیر به دست آمده از پوشه های train,valid را پرینت میکنیم.(این بخش صرفا جهت زیبایی و دریافت مقادیر فایل ها است و تاثیری در روند برنامه ندارد (از بخش های اصلی برنامه نیست))

```
print(f"train positive images: {len(os.listdir(train_pos))}")
print(f"train negative images: {len(os.listdir(train_neg))}")
print(f"valid positive images: {len(os.listdir(val_pos))}")
print(f"valid negative images: {len(os.listdir(val_neg))}")
```

```
train positive images: 599
train negative images: 673
valid positive images: 140
valid negative images: 148
```

تصاویر را به ۱۵۰*۱۵۰ پیکسل تبدیل میکنیم.با آزمون و خطا بهترین بازدهی را داد.

Bach_saze : مدل هارا دسته بندی میکند.مرتبط با کارت گرافیک است که با آزمون و خطا به دست می آید.(در اینجا ما از گرافیک 2080 RTX استفاده کردیم)

Epotchs : تعداد مرور دسته ها (چندین بار اجرا شد و بهترین بازده را در ۳۰ داشت)

مقدار valid , train باید نزدیک به هم باشد .اگر valid بزرگتر باشد مشکلی ندارد و جا دارد برای یادگیری بیشتر ولی اگر train بزرگتر باشد دچار مشکل میشویم.

مرحله نهم

نرمال سازی دیتا(Data Normalization)

```
train_generator = ImageDataGenerator(rescale=1./255)
valid_generator = ImageDataGenerator(rescale=1./255)
```

مرحله دهم

آماده سازی tain,valid از روی متغیرها

```
train_data_gen = train_generator.flow_from_directory(
    batch_size=batch_size, directory=train_dir, shuffle=True,
    target_size=(IMG_SIZE, IMG_SIZE),class_mode = 'binary'
)
```

```
Found 1272 images belonging to 2 classes.
```

```
val_data_gen = valid_generator.flow_from_directory(
    batch_size=batch_size, directory=valid_dir, shuffle=True,
    target_size=(IMG_SIZE, IMG_SIZE),class_mode = 'binary'
)
```

```
Found 288 images belonging to 2 classes.
```

مرحله یازدهم

مدل اصلی را از آدرس (لینک) زیردانلود و استفاده میکنیم

میتوان از مدل های دیگر نیز استفاده کرد

```
module_url = 'https://tfhub.dev/google/imagenet/inception_v3/classification/4'
```

مرحله دوازدهم

مقدار اصلی مدل ها را تعریف میکنیم

```
model = tf.keras.Sequential([
    hub.KerasLayer(module_url,
                   input_shape=(IMG_SIZE, IMG_SIZE, 3), trainable=False),
    tf.keras.layers.Dropout(rate=0.3),
    tf.keras.layers.Dense(1, activation='sigmoid',
                          kernel_regularizer=tf.keras.regularizers.l2(0.0001))
])
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
keras_layer (KerasLayer)     (None, 1001)              23853833
_________________________________________________________________
dropout (Dropout)            (None, 1001)              0
_________________________________________________________________
dense (Dense)                (None, 1)                 1002
=================================================================
Total params: 23,854,835
Trainable params: 1,002
Non-trainable params: 23,853,833
_________________________________________________________________
```

نشان دهنده خلاصه ای از پارامترهای مدل است.

مرحله سیزدهم

بخش train (مقادیر اصلی را که در بالا تنظیم کردیم ، استفاده میکنیم و شروع به train میکنیم. (بخش اصلی پروژه است و زمان زیادی برای اجرا وقت میبرد)

```
history = model.fit_generator(
    train_data_gen,
    steps_per_epoch=total_train // batch_size,
    epochs=epochs,
    validation_data=val_data_gen,
    validation_steps=total_val // batch_size,

)
```

```
Epoch 1/30
79/79 [==============================] - 15s 191ms/step - loss: 1.5493 - accuracy: 0.5334 - val_loss: 0.6634 - val_accuracy: 0.7257
Epoch 2/30
79/79 [==============================] - 11s 136ms/step - loss: 1.0504 - accuracy: 0.6330 - val_loss: 0.7101 - val_accuracy: 0.7049
Epoch 3/30
79/79 [==============================] - 11s 133ms/step - loss: 1.0333 - accuracy: 0.6218 - val_loss: 0.5309 - val_accuracy: 0.7431
Epoch 4/30
79/79 [==============================] - 11s 138ms/step - loss: 0.8643 - accuracy: 0.6688 - val_loss: 0.5237 - val_accuracy: 0.7535
Epoch 5/30
79/79 [==============================] - 11s 136ms/step - loss: 0.7979 - accuracy: 0.6975 - val_loss: 0.4982 - val_accuracy: 0.7986
Epoch 6/30
79/79 [==============================] - 11s 137ms/step - loss: 0.7674 - accuracy: 0.6990 - val_loss: 0.4751 - val_accuracy: 0.7847
Epoch 7/30
79/79 [==============================] - 11s 139ms/step - loss: 0.7280 - accuracy: 0.7062 - val_loss: 0.4770 - val_accuracy: 0.7882
Epoch 8/30
79/79 [==============================] - 11s 143ms/step - loss: 0.7099 - accuracy: 0.7014 - val_loss: 0.4783 - val_accuracy: 0.7812
Epoch 9/30
79/79 [==============================] - 14s 178ms/step - loss: 0.7093 - accuracy: 0.7014 - val_loss: 0.5528 - val_accuracy: 0.7778
Epoch 10/30
79/79 [==============================] - 11s 136ms/step - loss: 0.6554 - accuracy: 0.7269 - val_loss: 0.5312 - val_accuracy: 0.7431
Epoch 11/30
79/79 [==============================] - 11s 136ms/step - loss: 0.6497 - accuracy: 0.7404 - val_loss: 0.4922 - val_accuracy: 0.7812
Epoch 12/30
79/79 [==============================] - 11s 141ms/step - loss: 0.6442 - accuracy: 0.7365 - val_loss: 0.5409 - val_accuracy: 0.7743
Epoch 13/30
79/79 [==============================] - 11s 137ms/step - loss: 0.6380 - accuracy: 0.7182 - val_loss: 0.5077 - val_accuracy: 0.7639
Epoch 14/30
```

```
WARNING:tensorflow:From <ipython-input-24-0bb76eff41f2>:6: Model.fit_generator (from tensorflow.python.keras.engine.training) is deprecated and will be removed in a future version.
Instructions for updating:
Please use Model.fit, which supports generators.
Epoch 1/30
79/79 [==============================] - 15s 191ms/step - loss: 1.5493 - accuracy: 0.5334 - val_loss: 0.6634 - val_accuracy: 0.7257
Epoch 2/30
79/79 [==============================] - 11s 136ms/step - loss: 1.0504 - accuracy: 0.6330 - val_loss: 0.7101 - val_accuracy: 0.7049
Epoch 3/30
79/79 [==============================] - 11s 133ms/step - loss: 1.0333 - accuracy: 0.6218 - val_loss: 0.5309 - val_accuracy: 0.7431
Epoch 4/30
79/79 [==============================] - 11s 138ms/step - loss: 0.8643 - accuracy: 0.6688 - val_loss: 0.5237 - val_accuracy: 0.7535
Epoch 5/30
79/79 [==============================] - 11s 136ms/step - loss: 0.7979 - accuracy: 0.6975 - val_loss: 0.4982 - val_accuracy: 0.7986
Epoch 6/30
79/79 [==============================] - 11s 137ms/step - loss: 0.7674 - accuracy: 0.6990 - val_loss: 0.4751 - val_accuracy: 0.7847
Epoch 7/30
79/79 [==============================] - 11s 139ms/step - loss: 0.7280 - accuracy: 0.7062 - val_loss: 0.4770 - val_accuracy: 0.7882
Epoch 8/30
79/79 [==============================] - 11s 143ms/step - loss: 0.7099 - accuracy: 0.7014 - val_loss: 0.4783 - val_accuracy: 0.7812
Epoch 9/30
79/79 [==============================] - 14s 178ms/step - loss: 0.7093 - accuracy: 0.7014 - val_loss: 0.5528 - val_accuracy: 0.7778
Epoch 10/30
79/79 [==============================] - 11s 136ms/step - loss: 0.6554 - accuracy: 0.7269 - val_loss: 0.5312 - val_accuracy: 0.7431
Epoch 11/30
79/79 [==============================] - 11s 136ms/step - loss: 0.6497 - accuracy: 0.7404 - val_loss: 0.4922 - val_accuracy: 0.7812
Epoch 12/30
79/79 [==============================] - 11s 141ms/step - loss: 0.6442 - accuracy: 0.7365 - val_loss: 0.5409 - val_accuracy: 0.7743
Epoch 13/30
79/79 [==============================] - 11s 137ms/step - loss: 0.6380 - accuracy: 0.7182 - val_loss: 0.5077 - val_accuracy: 0.7639
Epoch 14/30
79/79 [==============================] - 11s 137ms/step - loss: 0.5882 - accuracy: 0.7428 - val_loss: 0.5048 - val_accuracy: 0.7778
Epoch 15/30
79/79 [==============================] - 11s 140ms/step - loss: 0.5963 - accuracy: 0.7412 - val_loss: 0.5414 - val_accuracy: 0.7604
Epoch 16/30
79/79 [==============================] - 11s 136ms/step - loss: 0.5623 - accuracy: 0.7524 - val_loss: 0.5556 - val_accuracy: 0.7708
Epoch 17/30
79/79 [==============================] - 13s 167ms/step - loss: 0.5621 - accuracy: 0.7436 - val_loss: 0.5017 - val_accuracy: 0.7743
Epoch 18/30
79/79 [==============================] - 11s 136ms/step - loss: 0.5475 - accuracy: 0.7516 - val_loss: 0.5385 - val_accuracy: 0.7326
Epoch 19/30
79/79 [==============================] - 11s 138ms/step - loss: 0.5999 - accuracy: 0.7492 - val_loss: 0.5413 - val_accuracy: 0.7674
Epoch 20/30
79/79 [==============================] - 11s 142ms/step - loss: 0.5449 - accuracy: 0.7436 - val_loss: 0.5148 - val_accuracy: 0.7778
Epoch 21/30
79/79 [==============================] - 11s 138ms/step - loss: 0.5822 - accuracy: 0.7373 - val_loss: 0.5204 - val_accuracy: 0.7674
Epoch 22/30
79/79 [==============================] - 11s 141ms/step - loss: 0.5349 - accuracy: 0.7564 - val_loss: 0.5248 - val_accuracy: 0.7604
Epoch 23/30
79/79 [==============================] - 11s 138ms/step - loss: 0.5560 - accuracy: 0.7532 - val_loss: 0.5437 - val_accuracy: 0.7500
Epoch 24/30
79/79 [==============================] - 11s 138ms/step - loss: 0.5514 - accuracy: 0.7516 - val_loss: 0.5345 - val_accuracy: 0.7361
Epoch 25/30
79/79 [==============================] - 13s 167ms/step - loss: 0.5331 - accuracy: 0.7596 - val_loss: 0.5125 - val_accuracy: 0.7535
Epoch 26/30
79/79 [==============================] - 11s 137ms/step - loss: 0.5666 - accuracy: 0.7349 - val_loss: 0.6057 - val_accuracy: 0.7361
Epoch 27/30
79/79 [==============================] - 11s 136ms/step - loss: 0.5564 - accuracy: 0.7484 - val_loss: 0.5253 - val_accuracy: 0.7465
Epoch 28/30
79/79 [==============================] - 11s 141ms/step - loss: 0.5294 - accuracy: 0.7532 - val_loss: 0.5612 - val_accuracy: 0.7604
Epoch 29/30
79/79 [==============================] - 11s 136ms/step - loss: 0.5549 - accuracy: 0.7428 - val_loss: 0.5233 - val_accuracy: 0.7778
Epoch 30/30
79/79 [==============================] - 11s 138ms/step - loss: 0.5455 - accuracy: 0.7580 - val_loss: 0.5496 - val_accuracy: 0.7500
```

مرحله چهاردهم

ذخیره کردن مدل

```
#saving model
model.save("/content/gdrive/My Drive/Kaggle/result_Humerus.h5")
```

مرحله پانزدهم

```
import json
his = history.history
with open("/content/gdrive/My Drive/Kaggle/history_Humerus.json", 'w') as file:
  json.dump(his, file)
```

پارامتر history **که هیستوری کامل از زمان** train **را نشان میدهد.**